# DISTRIBUTED COMPUTING ENVIRONMENTS FOR FUTURE SPACE CONTROL SYSTEMS



Pierre Viallefont

C.N.E.S
Toulouse - FRANCE

## ABSTRACT

The aim of this paper is to present the results of a CNES research project on distributed computing systems. The purpose of this research was to study the impact of the use of new computer technologies in the design and development of future space applications.

The first part of this study was a state-of-the-art review of distributed computing systems. One of the interesting ideas arising from this review is the concept of a "virtual computer" allowing the distributed hardware architecture to be hidden from a software application.

The "virtual computer" can improve system performance by adapting the best architecture (addition of computers) to the software application without having to modify its source code. This concept can also decrease the cost and obsolescence of the hardware architecture.

In order to verify the feasibility of the "virtual computer" concept, a prototype representative of a distributed space application is being developed independently of the hardware architecture.

Key Words: Distributed Computing, Distributed Architecture, Control Center.

## 1. OVERALL APPROACH

The motivation behind this research is the growing importance of distributed computing environments. First of all, a state-of-the-art review of distributed systems was made so as to reveal the main underlying concepts. We then determined what innovative contributions could be made to the design and development of our space computing applications by such distribution concepts. Once these contributions were clearly identified, it was decided to validate them by applying them to the development of a prototype representative of a distributed space application.

## 2. STATE-OF-THE-ART REVIEW

### 2.1 Concepts

#### 2.1.1 Definition

There are many definitions of distributed systems. We chose the following one: "A distributed system is a system whose behaviour is determined by algorithms specifically designed to take into account and use several processing places".

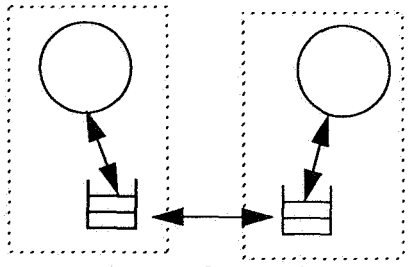
#### 2.1.2 Client/server Model

The customer/server model is certainly the most widespread concept in the literature dealing with distributed architectures. The server defines services it makes available to the client. The client can access the server only through the set of services (functions) the server has decided to export. The server can serve several clients.

#### 2.1.3 Distributed programming techniques

There are two categories of distributed programming techniques:

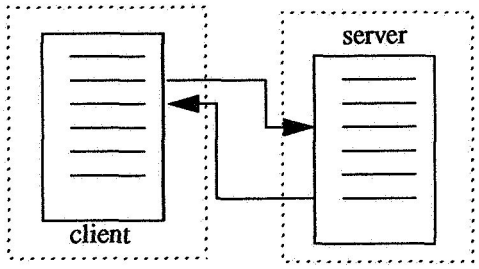
--> Inter Process Communication (IPC) allowing two remote processes to communicate by sending messages to each other. TCP/IP sockets are a good example of this.



Inter Process Communication

--> Remote Procedure Calls (RPC) allowing two remote processes to communicate in a different way, namely through the transmission of parameters. The

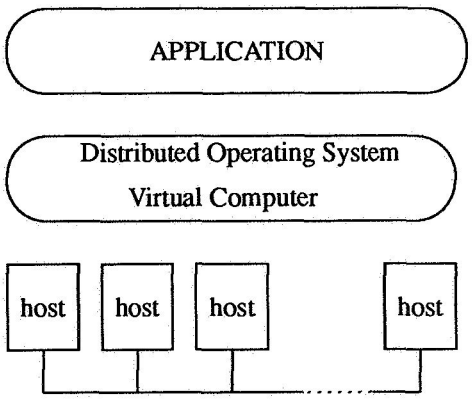
customer/server model can be implemented quite simply using RPCs. The customer calls a server function using an RPC and then waits for the answer. The server carries out the processing and returns the results to the customer. One example of the use of RPCs is that of the SUN RPCs used to build the NFS distributed file system (Ref. 1).




Remote Procedure Call

2.1.4 TheVirtual Computer

Another concept highlighted in the state-of-the-art review is that of the virtual computer. This approach allows the hardware architecture to be masked to an application in order to give it the impression that it is being run on a centralized system.




Virtual Computer

The virtual computer concept is closely linked with the concept of transparency. Several transparency levels are defined in the ANSA project (Ref. 2):

--> access-to-object transparency: an object (such as a file) may be accessed (i.e. opened, read, deleted etc.) in the same way whether locally or remotely. An example of a system offering access-to-file transparency is the NFS (Network File System). However, a real distributed system must provide this transparency for all the objects it manages (not only files, but also peripherals, processes, memory etc.).

--> location transparency: a user or an application need not worry about the location of the objects he/it is handling. The NFS also offers this type of transparency: nothing in the filenames indicates the location of these objects.

--> concurrency transparency: several users or applications may share a remote object without being aware of it.

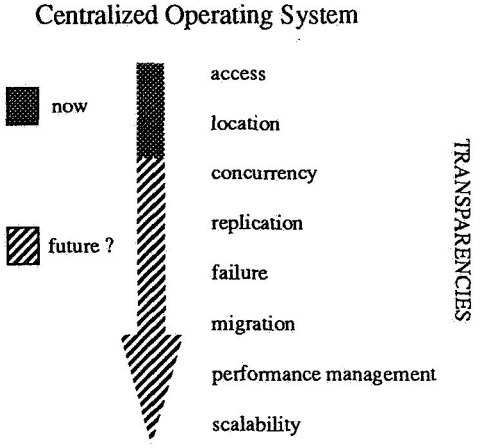
--> replication transparency: some objects are replicated without the application being aware of it. This is very useful for implementing hardware fault tolerance techniques by process replication.

--> failure transparency: the occurrence of faults is masked to applications, or at least the work in progress is completed.

--> migration transparency: objects can migrate from one computer to another without the application being aware of it.

--> performance management transparency: the system can reconfigure itself dynamically in order to improve performance in a transparent manner.

--> scaling transparency: the system or applications can change the execution scale (e.g.: increased number of computers in a network) without having to change the algorithms.

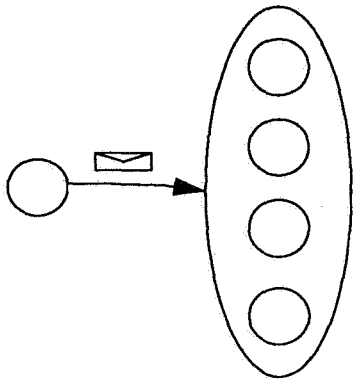



Distributed Operating System
(virtual computer)

The implementation of the different transparency levels can be used to define a real distributed operating system based on the concept of a virtual computer. Most Industry or Research products provide both access and location transparency. Some provide even more, but at this point in time, none of the systems investigated are able to provide all the different kinds of transparency mentioned above.

2.1.5 Process groups

This concept can be found in many of the systems investigated. A process group, as its name indicates, groups together several different processes. Its advantage is that all the processes belonging to the same group receive the same messages.



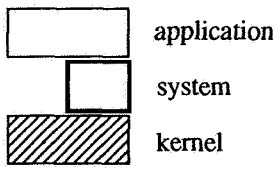
Process Groups

With this concept, message broadcast and above all fault tolerance can easily be implemented by replicating the same processes on different sites.

2.2 Systems investigated

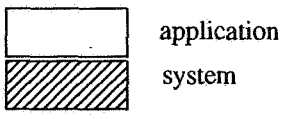
Having examined both Research and Industry products, three types of system could be identified:

--> native systems integrating distribution at kernel level. The operating system is built over the kernel. The most technologically advanced kernels at present are Chorus (Ref. 3), Mach (Ref. 4) and Amoeba (Ref. 5).



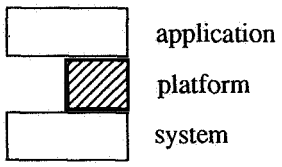

Native Systems

--> integrated systems implementing functions related to distribution.




Integrated Systems

--> platforms or toolboxes located over the operating system. They provide users with a distributed environment without masking the operating system. The most advanced are ISIS (Ref. 6), ANSA (Ref. 2), DELTA-4 (Ref. 7) and OSF/DCE (Ref. 8).




Platforms

All current state-of-the-art systems provide at least access and location transparency. Some of them offer replication transparency (like ISIS), and others fault transparency (like DELTA-4). All these systems are oriented towards use with Unix.

With respect to standardization, the OSF/DCE system would appear to be the most promising as it is supported by the majority of Unix manufacturers. Unfortunately, OSF/DCE is not yet available and will definitely not be available before 1993.

## 3. THE CONTRIBUTION OF THE CONCEPTS

3.1 The virtual computer

3.1.1 Adapting the architecture to the application

When building a spacecraft control center, the manufacturer and hardware architecture are both selected at the outset of the project, before software development begins. Sometimes, however, software development can last several years (1 to 5).

One never knows before the application's validation if the hardware architecture will be efficient enough to run the application. If it is not efficient enough, the current configuration either has to be upgraded (through extra memory, CPU board, additional disks etc.), or the software code has to be optimized. If the

power of the CPU board cannot be upgraded, one or more extra computers have to be added, requiring a change in the architecture, and therefore a change in the application code.

Using a distributed system implementing the concept of a virtual computer enables the architecture's distribution to be masked to the application. Thus, it is quite possible on integration to redistribute application components over another distributed architecture while maintaining its performance standards (addition of a computer) and without having to change the application code.

3.1.2 Keeping one step ahead of architecture obsolescence

Owing to the current developments in data processing technologies, the price/performance ratio of computers is constantly decreasing such that the architecture selected at the beginning of the project is technically superseded by the end, and the resulting price/performance ratio is very poor. The project investment cost may appear to be relatively high compared with the architecture's real value at the time of validation. Furthermore, there may be a better architecture/manufacturer pair at the time of the application's validation.

One solution consists in choosing the target architecture after the application has been developed. Firstly, this requires the use of a standard (Unix) operating system in order to be independent of the manufacturer. If this system implements the virtual computer concept, the application also becomes independent of the architecture. However, this poses many problems: firstly, if no architecture is chosen, on which computers will the application be developed ? This problem can be solved by buying "low-quality" workstations which will be used exclusively for development work. Secondly, is it really possible to do without the specific characteristics of a project (communication protocols, fault tolerance etc.) which often determine the choice of architecture at the outset of the project ?

3.2 The client/server model

3.2.1 Simplifying the development of distributed applications

The design and development of a distributed application is quite complex: using tools such as TCP/IP sockets is not easy, and the final development of a distributed application may even be distinctly difficult (error reproducibility difficulties, no final development tools etc.).

The development of distributed applications can be simplified using the RPC-based client/server model. RPCs provide interface description languages and generators which allow the developer to concentrate exclusively on the development of server and customer functions without worrying about network communication. The server interfaces are clearly defined and, as a result, their final development becomes easier.

3.3 Process groups

3.3.1 Tolerating hardware faults

Hardware fault tolerance acts as a brake upon distribution. It has a direct effect on both the architecture's design (redundant computers) and development (reconfiguration scenarios, failure processing etc.).

In some applications the problem is solved by:

--> using fault-tolerant computers (Tandem, Stratus etc.)

--> replicating computers so as to be able to reinitiate the application on the redundant computers.

The solution based on fault-tolerant computers may be deemed expensive. Computer replication may be inappropriate as it requires that the application should be stopped and then reinitiated on another machine with the same hardware configuration.

Fault tolerance problems can be solved efficiently - and without having to buy specific computers - with a distributed system implementing the process groups on replicated computers. Indeed, as the processes are replicated on distinct computers, the failure of a single computer does not affect the application's operation.

## 4. THE PROTOTYPE

4.1 Objectives

The objective of the prototype's development was the practical validation of the aforementioned concepts, namely the concept of a virtual computer, the client/server model and process groups.

Thus, the prototype was designed:

--> to be independent of hardware architecture. It is hoped that our application will be able to run on one,

two or "x" number of computers without having to resort to recompilation. It was therefore decided to develop and validate our distributed application on a single computer before testing it on a distributed architecture.

--> to be independent of the manufacturer by using Unix standards (portability).

--> to tolerate computer hardware failures without affecting the application's operation.

4.2 Functional characteristics

It was decided to put the previous concepts into practice in an application typical of the space environment, and building a prototype inspired by spacecraft control centers appeared a judicious idea.

The following functions were chosen:

--> Telemetry acquisition and decommutation,
--> Real-time telemetry monitoring,
--> Control and monitoring of the distributed application,
--> Real-time logbook,
--> Off-line analysis of the logbook,
--> Off-line telemetry processing.

4.3 Development environment

The development environment of the prototype consists of:

--> the ISIS toolbox developed by Cornell University, to manage fault tolerance by using the process group concept.
--> the EASY RPC product developed by the French company "Cap Gemini Innovation". This product, based on Sun RPCs, also provides both access and location transparency; it therefore enables an application to be developed independently of the architecture (access and location transparency). The EASY RPC product enables easy implementation of the client/server model, thus facilitating the programming and final development of the distributed application.

--> a Unix system complying with POSIX.1 and XPG3 standards, to be independent of the manufacturer. Whilst our application is intended to be developed on a SUN4 workstation, the target architecture will actually comprise DEC, HP and SUN Unix computers.

--> the C++ language for modular software development.

--> the OSF/Motif system for multiwindowing combined with a graphic interface generator.

## 5. CONCLUSION

This research project is not yet completed, since the prototype is still being developed. Nevertheless, positive results are expected and it is hoped that future developments in spacecraft control centers will integrate distribution concepts so as to be able to develop space applications which are flexible, upgradeable, hardware fault tolerant and above all independent of hardware architecture.

## 6. REFERENCES

1. R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, B. Lyone. 1985. "Design and implementation of the Sun Network File System". In Usenix proceedings.

2. Advanced Networked Systems Architecture. 1989. "An Engineer's Introduction to the Architecture". Architecture Projects Management Limited.

3. MP. Rouille, P. Viallefont. 1990. "Evaluation du systeme d'exploitation reparti et temps reel Chorus". Rapport de stage CNES - RA/TE/IS/MIS/SM.

4. Tevanian, Avadis, Rashid, Richard. 1987. "MACH: A basis for Future UNIX Development". Department of Computer Science, Carnegie Mellon University, Pittsburgh, Technical report CMU-CS-87-139

5. Mullender and all. 1989. "Amoeba - High Performance Distributed Computing". In Technical Report CWI, CS-R8929.

6. K. Birman, T. Joseph. 1988. "The ISIS System Manual". Cornell University, Ithaca, NY.

7. D. Powell, P. Verissimo, G. Bonn, F. Waeselynck, D. Seaton. 1988. "The Delta-4 Approach to Dependability in Open Distributed Computing Systems". In proceedings 18th Int. Symposium on Fault-tolerant Computing- IEEE.

8. Open Software Foundation. 1990. "Distributed Computing Environment- Overview and rationale".